TMDB Box Office Prediction

Can you predict a movie's worldwide box office revenue?

Prepared By: Soumee Ghosh (2211444)

Dataset Description:

In this dataset, we are provided with 7398 movies and a variety of metadata obtained from The Movie Database (TMDB). Movies are labelled with id. Data points include cast, crew, plot keywords, budget, posters, release dates, languages, production companies, and countries.

We are predicting the worldwide revenue for 4398 movies in the test file.

Note - many movies are remade over the years, therefore it may seem like multiple instance of a movie may appear in the data, however they are different and should be considered separate movies. In addition, some movies may share a title, but be entirely unrelated.

E.g. The Karate Kid (id: 5266) was released in 1986, while a clearly (or maybe just subjectively) inferior remake (id: 1987) was released in 2010.

Also, while the Frozen (id: 5295) released by Disney in 2013 may be the household name, don't forget about the less-popular Frozen (id: 139) released three years earlier about skiers who are stranded on a chairlift.

The analysis can be done in the following way:

- Understanding the data
- EDA and Data Cleaning
- Data Pre-Processing
- Model building & Evaluation
- Hyperparameter Tuning
- Prediction.

A: Understanding the data

Firstly, we import the necessary libraries for performing the prediction. WEeload the training and the test data. We thereafter try to check the first 5 observations of the dataframe using .head() method.

We then try to look for the datatypes, null values and statistical decription of the dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     3000 non-null   int64
 1   belongs_to_collection  604 non-null    object
 2   budget                 3000 non-null   int64
 3   genres                 2993 non-null   object
 4   homepage               946 non-null    object
 5   imdb_id                3000 non-null   object
 6   original_language      3000 non-null   object
 7   original_title         3000 non-null   object
 8   overview               2992 non-null   object
 9   popularity             3000 non-null   float64
 10  poster_path            2999 non-null   object
 11  production_companies   2844 non-null   object
 12  production_countries   2945 non-null   object
 13  release_date           3000 non-null   object
 14  runtime                2998 non-null   float64
 15  spoken_languages       2980 non-null   object
 16  status                 3000 non-null   object
 17  tagline                2403 non-null   object
 18  title                  3000 non-null   object
 19  Keywords               2724 non-null   object
 20  cast                   2987 non-null   object
 21  crew                   2984 non-null   object
 22  revenue                3000 non-null   int64
dtypes: float64(2), int64(3), object(18)
memory usage: 539.2+ KB
```

We observe that there are total 3000 entries and data types of the features consists of float, integer and object.

There are also presence of null values in the dataset both in the training and testing data.

B: EDA and Data Cleaning

So our next step should be dealing with the null values through imputing.

Handling missing values:

At first we try to deal with the single missing value for the release date feature and replace it with the original release data (found through web search)

```
3]: # Addin the release date 05/01/2020, which I found through a quick online search
    test.loc[test['release_date'].isnull()==True, 'release_date']= '5/1/00'
    test[test["release_date"]== '5/1/00']
```

Now to deal with the missing values for the nominal data we fill them with "none" whereas for numerical data we replace the missing values with the mean value.

Next, we move to formatting the dates and creating new features i.e. we perform feature extraction from a single column named feature.

From the release date feature we extract release year, day, month for both the training and testing data.

Now a point to note : Since the Kaggle competition was in 2019 so there shouldn't be any release date after 2019.

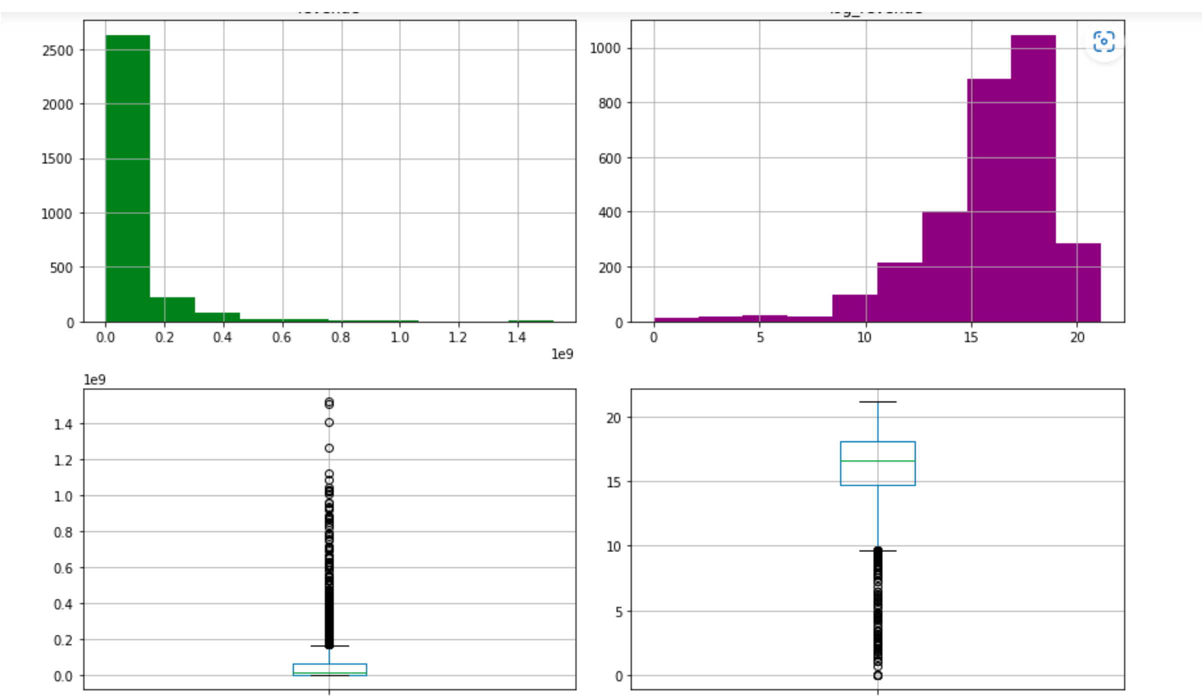
So we fix the dates.

```
In [23]: # Fixing the dates
         def fix_date(x):
             if x > 2019:
                 return x - 100
             else:
                 return x

         train['release_year'] = train['release_year'].apply(lambda x: fix_date(x))
         test['release_year'] = test['release_year'].apply(lambda x: fix_date(x))
```

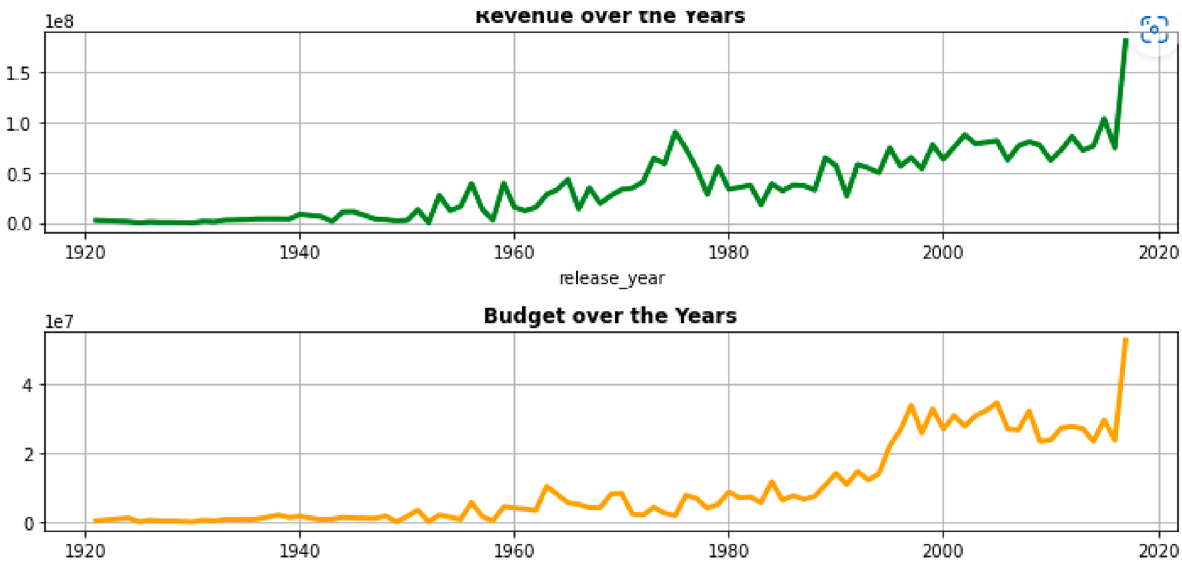
Now we perform the Univariate Analysis:

We convert the revenue into log of revenue and look into its distribution.

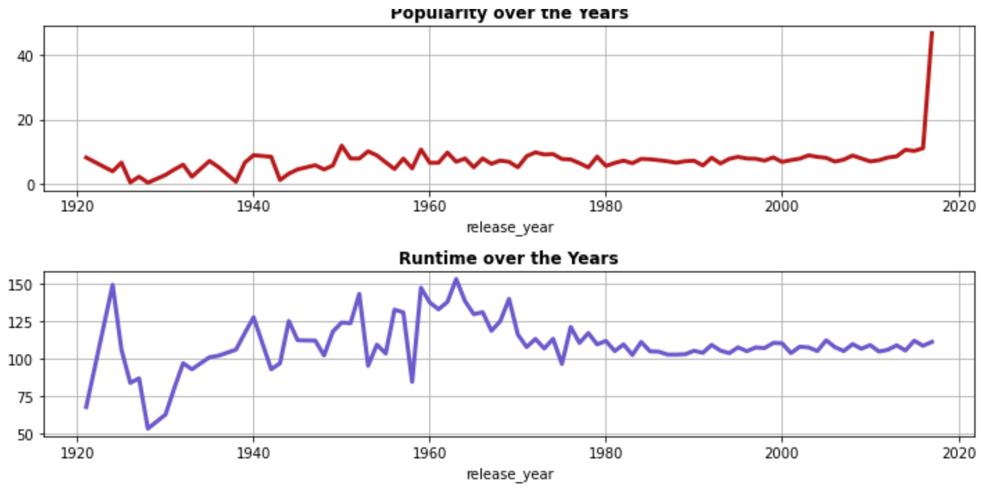


Then we perform univariate analysis for all the categorical columns i.e. budget, popularity, runtime.

Then we try to observe the revenue, budget, popularity and runtime over the years.



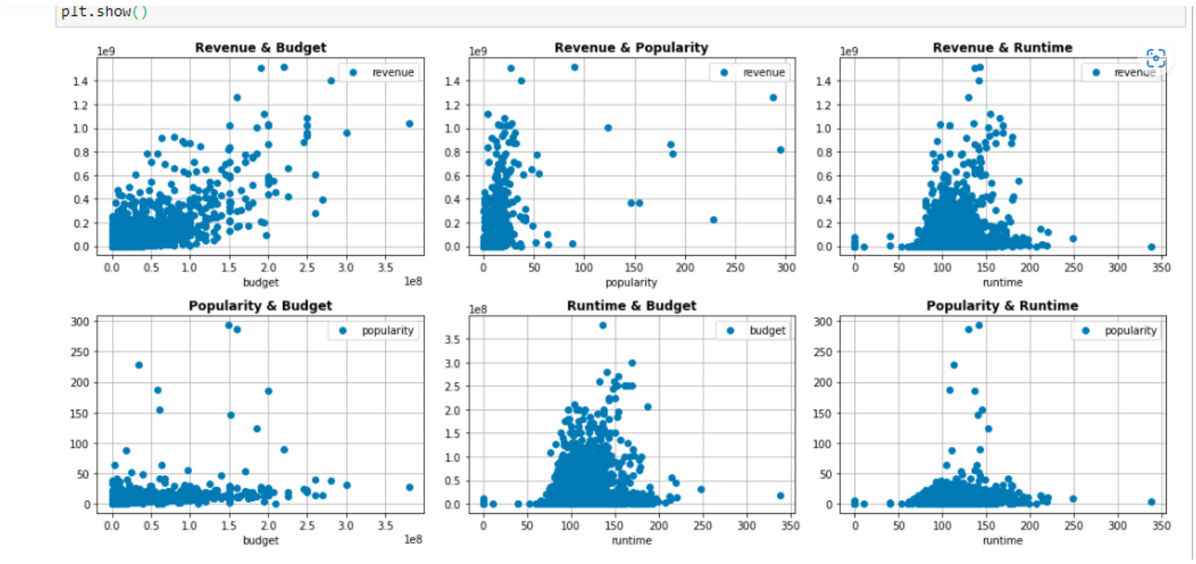
We can see that revenue and budget are rapidly increasing over the years.



For popularity, we can see that there has been a constant popularity for the movies over the years with a sharp increase towards the 2018-19 period.

In case of runtime we see that it has decreased since 1980 onwards and has been steadily decreasing since then.

Here is a compact comparison between revenue and other categorical columns :



Next we try to extract certain features by counting their occurrences like we count the genres, spoken language count, cast count and crew count as this might have an effect on the revenue of the movie.

Thereafter we convert the categorical data into numerical data for our model building using .cat.codes.

We observe that for certain movies , the budget and runtime column have value as zero which is absurd so we impute them with mean value.

C: Data pre-processing

Now we move into the model building section where we assign the data corresponding to the target and predictor variables and we split the training data into train and validation data.

D: Model building & Evaluation

We firstly perform regression using models:

- Random Forest
- XG Boost model

Random Forest model:

```
RandomForestRegressor
RandomForestRegressor(random_state=1)
```

```python
# Prediction
y_pred_rf = rf_model.predict(X_valid_full)
```

```python
# Calculate MAE
mae_rf = mean_absolute_error(y_pred_rf, y_valid)

print("Mean Absolute Error RF:" , mae_rf)
```

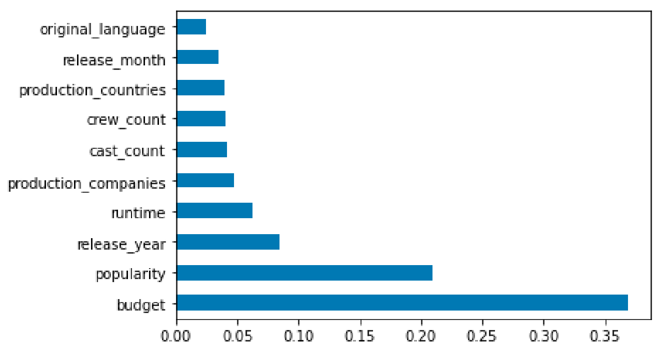
```
Mean Absolute Error RF: 1.356911847183321
```

We find that the mean absolute error is 1.3569.

Thereafter we try to calculate the feature importance:

```python
]: # Calculating feature importance
feat_importances = pd.Series(rf_model.feature_importances_, index=X_train_full.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1e884c5b550>
```



Here we keep the top 10 features.

XG Boost model:

```
xgb_model.fit(X_train_full, y_train)
```

0]:

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...)
```

2]:

```
# Prediction
y_pred_xgb = xgb_model.predict(X_valid_full)
```

3]:

```
# Calculate MAE
mae_xgb = mean_absolute_error(y_pred_xgb, y_valid)

print("Mean Absolute Error XGBOOST:" , mae_xgb)
```

```
Mean Absolute Error XGBOOST: 1.5031416871680505
```

Here the mean absolute error is 1.503 .

Thus we see that in terms of mean absolute error our random forest model performs better as compared to the XG boost model.

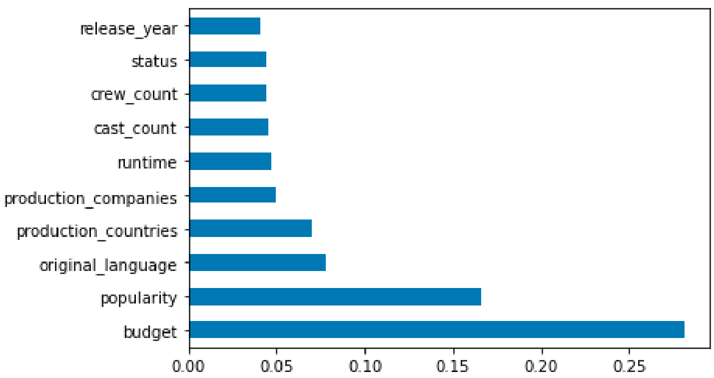
Now we calculate the feature importance using XG boost model:

Mean Absolute Error XGBOOST: 1.5031416871680505

4]:

```
# Calculating feature importance for the XGBoost Model
feat_importances = pd.Series(xgb_model.feature_importances_, index=X_train_full.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

4]: `<matplotlib.axes._subplots.AxesSubplot at 0x1e8848faac0>`



D: Hyperparameter Tuning

Finally, we try to tune our model for better results.

Random Forest:

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best parameters: {'n_estimators': 150, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_depth': 25}
Best score: 0.5134566363474853
Validation mse: 4.0805722085321126
```

XGBoost Model:

```
Best hyperparameters:  {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100, 'reg_alpha': 1}
Best score:  4.661748536646262
validation score:  4.04106184119158
```

Catboost Model:

```
982:	learn: 1.7060156	total: 14.2s	remaining: 245ms
983:	learn: 1.7059573	total: 14.3s	remaining: 232ms
984:	learn: 1.7055220	total: 14.3s	remaining: 218ms
985:	learn: 1.7054293	total: 14.4s	remaining: 204ms
986:	learn: 1.7053712	total: 14.4s	remaining: 190ms
987:	learn: 1.7051113	total: 14.4s	remaining: 175ms
988:	learn: 1.7048084	total: 14.5s	remaining: 161ms
989:	learn: 1.7045039	total: 14.6s	remaining: 147ms
990:	learn: 1.7037906	total: 14.6s	remaining: 133ms
991:	learn: 1.7034651	total: 14.7s	remaining: 118ms
992:	learn: 1.7028766	total: 14.7s	remaining: 104ms
993:	learn: 1.7020148	total: 14.7s	remaining: 88.9ms
994:	learn: 1.7013160	total: 14.8s	remaining: 74.1ms
995:	learn: 1.7007287	total: 14.8s	remaining: 59.3ms
996:	learn: 1.7006505	total: 14.8s	remaining: 44.5ms
997:	learn: 1.7005926	total: 14.8s	remaining: 29.7ms
998:	learn: 1.7004019	total: 14.9s	remaining: 14.9ms
999:	learn: 1.7003504	total: 14.9s	remaining: 0us
validation score:  4.005321556779982
```

Thus comparing the validation scores of the three models we find that catboost model performs the best on our data.

Predictions:

We use the catboost model which is hyperparameter tuned as our final model and perform predictions on the test data.